
REACTIVE
PROGRAMMING

A PRACTICAL INTRODUCTION

Open West 2016

by Seth House
@whiteinge

https://github.com/whiteinge/presentations/tree/master/openwest_2016-07_reactive-programming

WHAT IS REACTIVE?

- Declarative.
- React to an event.
- A unified API for sync & async operations.

REACTIVE EXTENSIONS

http://reactivex.io/

LEARNING RX

COMMON API

It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.

– Alan Perlis

RX API

- Large API, but...
- Filtering: `filter`
- Transforming: `map`, `reduce`
- Collecting: `scan`
- Buffering: `take`, `takeLast`, `pauseable`/`pauseableBuffered`
- Combining: `merge`/`concat`, `flatMap`/`concatMap`, `combineLatest`

PRIMITIVES

- Observer.
- Observable.
- Subscriptions.
- Disposables.

LANGUAGE IMPLEMENTATIONS

C#, C# (Unity), C++, Clojure, Groovy, JRuby, Java, JavaScript, Kotlin, Python, Ruby, Scala, and Swift.

Platform-specific support for Android, Cocoa, and Netty.

SHORT EXAMPLES

AJAX REQUEST

```
var mySubscription = Rx.DOM.get('https://api.github.com/users')
    .subscribe(response => console.log('Got response', response));
```

DOM EVENTS

```
var domEl = document.querySelector('#thelink');
var clicks = Rx.Observable.fromEvent(el, 'click')
    .scan(acc => acc + 1, 0)
    .subscribe(count => console.log(`Seen ${count} clicks.`));
```

SERVER-SENT EVENTS

```
var source = Rx.DOM.fromEventSource("/events");
source.map(JSON.parse).subscribe(
    msg => console.log('msg', msg),
    err => console.log('Stream complete.'));
```

POLLING

```
var results = Rx.Observable.interval(20000)
    .flatMapLatest(() => Rx.DOM.get('https://api.github.com/users'))
    .distinctUntilChanged()
    .subscribe(x => console.log('New results: ', x));
```

COMBINE AJAX REQUESTS

```
var combinedResults = Rx.DOM.getJSON('https://api.github.com/users/1'
    .flatMap(user_resp => Rx.DOM.getJSON(user_resp.followers_url)
        .map(followers => ({user: user_resp, followers}));
    });
```

IMPLEMENTATION

OBSERVER

- A consumer.
- Optional next, error, and completed methods.

```
var myObserver = {
    onNext: x => console.log('Got value', x),
    onError: err => console.log('Got error', err),
    onCompleted: () => console.log('Completed'),
};
```

OBSERVABLE

- A function that takes an observer and returns a cancellation function.
- Glue to connect a producer to a consumer (observer).

```
function myObservable(observer) {
    setTimeout(() => observer.onNext('Hello'), 1000);
    setTimeout(() => observer.onCompleted(), 2000);

    return function() {
        console.log('Canceled.');
    };
}
```

COMMON OBSERVABLE SOURCES

`just`, `from`, `range`, `fromEvent`, `fromEventPattern`, `fromCallback`, `fromNodeCallback`, `fromPromise`, `generate`, `generateWithAbsoluteTime`, `generateWithRelativeTime`

SUBSCRIPTIONS AND DISPOSABLES

Start Listening:

```
var mySubscription = myObservable.subscribe(myObserver);
```

Stop Listening:

```
mySubscription.dispose();
```

Stop Listening Automatically:

```
mySubscription.take(3);
```

COLD VS. HOT

- Movie vs. live performance.

OPERATORS

```
function map(source, projectionFn) {
    return new Observable(function(observer) {
        var mapObserver = {
            onNext: x => observer.next(projectionFn(x)),
            onError: err => observer.onError(err),
            onCompleted: observer.onCompleted(),
        };

        return source.subscribe(mapObserver);
    });
}
```

UNICAST VS. MULTICAST

- Reuse & share the underlying subscription, or
- Subscribe individually.

USE RX

USE DESCISION TREES

- Which RxJS creation operator?
- Which RxJS instance operator?

MERGE OBSERVABLES

Example questions to ask when combining Ajax requests:

- Output each response as soon as it comes in?
- Output each response in the same order as the request?
- Wait for both to complete and combine them?
- Do one request, and use the result within the next?

DEBUGGING

```
myObservable
    .filter(x => x.someAttr)
    .do(x => console.log('Passed the filter: ', x))
    .map(doSomethingWithSomeAttr)
    .do(x => console.log('Current value of x: ', x));
```

LONG EXAMPLES

CLOSE A MODAL

```
var keysource = Rx.dom.keydown(window)
    .pluck('keycode')
    .filter(x => x === 27); // escape key

var clicksource = Rx.dom.click(window)
    .skip(1) // ignore first click that opens the modal
    .filter(ev => document.querySelector(this).contains(ev.target));

keysource.merge(clicksource)
    .take(1)
    .subscribe(closeModal);
```

TRACK APPLICATION STATE

React's Flux implemented in RxJS. Full example.

```
var Dispatcher = new Rx.Subject();

var Store1 = Dispatcher
    .filter(x => x.tag === 'foo')
    .scan(collectUserInput)
    .flatMap(ajaxRequest);

var View1 = Store1
    .map(formatResponseAsHTML);

var contentEl = document.querySelector('#content');
var renderer = View1.subscribe(function(content) {
    React.render(content, contentEl);
});
```

SERVE HTTP REQUESTS

Use Node.js's builtin HTTP module. Full example.

```
var requests$ = createServer(8000).share();

var handleSomePath = requests$
    .filter(x => return x.req.url === '/somepath')
    [...];

function createServer(port) {
    return Rx.Observable.create(function(observer) {
        var server = http.createServer(function(req, resp) {
            observer.onNext({req: req, resp: resp});
        });

        server.listen(port);

        // Disposing the server observable will stop the server.
        return function() {
```

LISTEN TO SALT'S EVENT BUS

Using RxPy. (Watch for Salt PR.)

```
def salt_events_observable(observer):
    log.debug('Starting Salt event listener.')
    event_bus = salt.utils.event.get_event(...)

    def deserialize_and_emit(raw):
        mtag, data = salt.utils.event.SaltEvent.unpack(raw)
        observer.on_next({'tag': mtag, 'data': data})

    def destroy():
        log.debug('Destroying Salt event listener.')
        event_bus.destroy()

    event_bus.set_event_handler(deserialize_and_emit)
    return destroy

source = Observable.create(salt_events_observable).publish().ref_cour
```

RESOURCES

SANDBOX

```
data:text/html,<!doctype html><html><script src="https://cdnjs.cloudf
```

LINKS

API Docs:

- Operators documentation
- The Big List (TM) of RxJS operators

Decision Trees:

- Which RxJS creation operator?
- Which RxJS instance operator?
- Broad Rx Decision Tree

Beginner resources:

- RxJS Koans
- The Rx Book

Advanced resources:

- Building Observables